VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

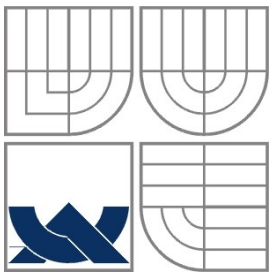
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

GENEROVÁNÍ PROCEDURÁLNÍCH TERÉNŮ

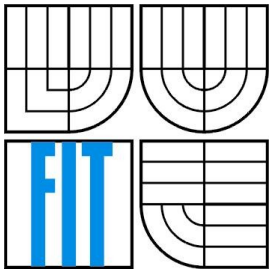
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

MICHAEL PETLAN



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

GENEROVÁNÍ PROCEDURÁLNÍCH TERÉNŮ
GENERATING PROCEDURAL TERRAINS

BAKALÁŘSKÁ PRÁCE
BACHELOR‘S THESIS

AUTOR PRÁCE — MICHAEL PETLAN
AUTHOR

VEDOUCÍ PRÁCE — ING. JAROSLAV PŘIBYL
SUPERVISOR

Abstrakt

Tato bakalářská práce se věnuje problematice algoritmů náhodného generování výškových map. Popisuje základní teoretické poznatky a rozvíjí je o využití L-systémů pro vytváření říční sítě. V rámci práce byla implementována knihovna algoritmů pro generování výškových map a jednoduchá aplikace, demonstrující její použití.

Abstract

This thesis deals with algorithms of stochastic generating of height maps. The thesis describes the basic theoretical knowledge and extends it by usage of L-systems for generating a network of rivers. There has been implemented a library of the height map generating algorithms and a demo application within the thesis.

Klíčová slova

výšková mapa, midpoint, Perlinův šum, box filtr, hillgrow, L-systém

Keywords

height map, diamond-square, Perlin noise, box filter, hillgrow, L-system

Citace

PETLAN, Michael. *Generování procedurálních terénů*. 2012. Bakalářská práce. FIT VUT v Brně.

Generování procedurálních terénů

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Jaroslava Přibyla.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Michael Petlan
16.5.2012

Poděkování

Děkuji vedoucímu Ing. Jaroslavu Přibylovi, který mi poskytoval rady a odbornou pomoc.

Obsah

1 Úvod

Tato bakalářská práce se věnuje problematice algoritmů náhodného generování výškových map. Popisuje základní teoretické poznatky a rozvíjí je o využití L-systémů pro vytváření říční sítě. V rámci práce byla implementována knihovna algoritmů pro generování výškových map a jednoduchá aplikace, demonstrující její použití.

Zde se setkáváme s pojmem *výšková mapa* [1]. Jedná se o dvourozměrnou matici hodnot, které určují výšku povrchu v daném bodě. Je to tedy rastrová reprezentace. Z popisu je jasné, že pomocí výškové mapy lze vyjádřit pouze tvar povrchu krajiny. Nelze pomocí ní tedy popsat jeskyně, tunely, převisy a podobné útvary.

My se blíže zaměříme na metody *procedurálního generování výškových map*. Pod tímto pojmem rozumíme algoritmy, vytvářející náhodně zvrásněné povrchy, většinou reprezentované právě výškovou mapou, kterou pak můžeme dále zpracovávat či zobrazovat. Pokusíme se dosáhnout co největší realističnosti generovaných map.

1.1 Osnova práce

V první kapitole se zběžně seznámíme s některými dostupnými nástroji, které s výškovými mapami pracují.

Druhá kapitola se věnuje popisu základních algoritmů generování a úpravy výškových map, jejich matematickému pozadí a problému vlivu náhodnosti v jednotlivých algoritmech.

Třetí kapitola nastiňuje nové postupy a možnosti, jak přizpůsobit výsledný povrch reálné krajině.

Ve čtvrté a páté kapitole je popsán koncept a implementace v rámci této práce vytvořené C++ knihovny a testovací aplikace, demonstrující použití knihovny.

Závěrečné kapitoly jsou zaměřeny na možnosti testování a dalšího vývoje projektu.

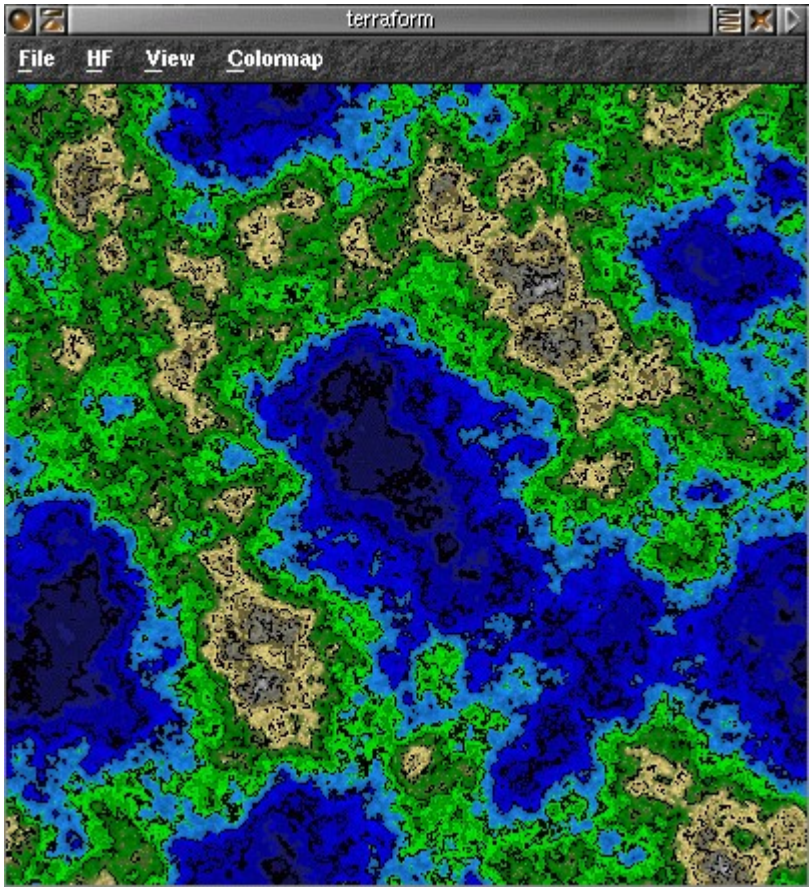
1.2 Existující nástroje pro práci s terény

Doposud bylo vyvinuto mnoho nástrojů, pracujících s výškovými mapami a využívajících metody procedurálního generování. Některé jsou pouze generátory, jiné umožňují i různé způsoby uživatelské editace, od změn parametrů generování v reálném čase až po prakticky "ruční" modelování. Několik z nich si zde stručně představíme.

1.2.1 Terraform [2]

- generátor
- opensource, již ukončený projekt
- Linux/Unix, Gtk+

Jednoduchý generátor výškových map, nabízí několik algoritmů. Program je možno ovládat pomocí příkazového řádku nebo grafického rozhraní umožňující i 3D náhled. Vygenerované mapy jsou ukládány do několika obrázkových formátů.



Obrázek 1: Screenshot z programu Terraform

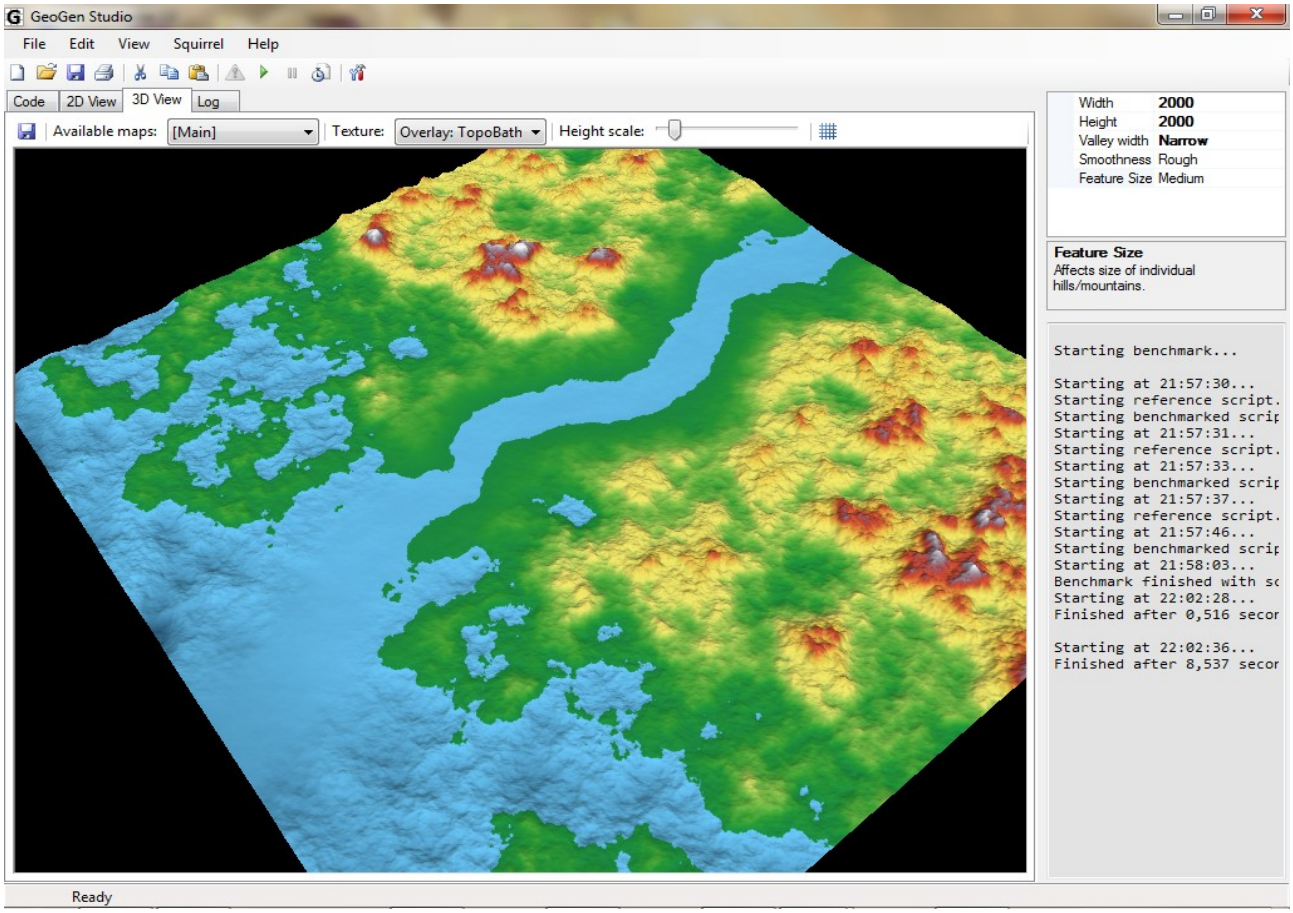
1.2.2 Height Map Editor [3]

- generátor a editor
- opensource – GNU GPL
- Linux/Unix, MS Windows, SDL

Program zaměřený jako co nejjednodušší editor výškových map. Kromě generování pseudonáhodného povrchu umožňuje snižovat nebo naopak zvyšovat uživatelem zadaná místa, vyhlazovací filtry, otáčení, přidávání vlastních útvarů a další. Pro zobrazení editované mapy program používá 2D pohled, kde je výška vyjádřena barvou. Mapy lze uložit buď ve standardním rastrovém formátu BMP (24bitová barevná hloubka – 256 odstínů šedi) nebo vlastním HMP, který je navržen přímo autorem.

1.2.3 GeoGen [4]

- generátor
- opensource
- MS Windows



Obrázek 2: Okno programu GeoGen; karty nabízí přepínání mezi různými pohledy

GeoGen zasluhuje pozornost zejména proto, že generuje terén podle uživatelem zadaného skriptu (*Squirrel scripting language* [5]), který definuje parametry, pořadí a způsob kombinace jednotlivých operací generovacího procesu. Každá operace vygeneruje vlastní výškovou mapu a ty se pak mohou za použití maskování sčítat, násobit nebo jinak prolínat. Nabízí se tak velmi široké spektrum možností, jak dosáhnout požadovaného terénu a jak s ním dále experimentovat.

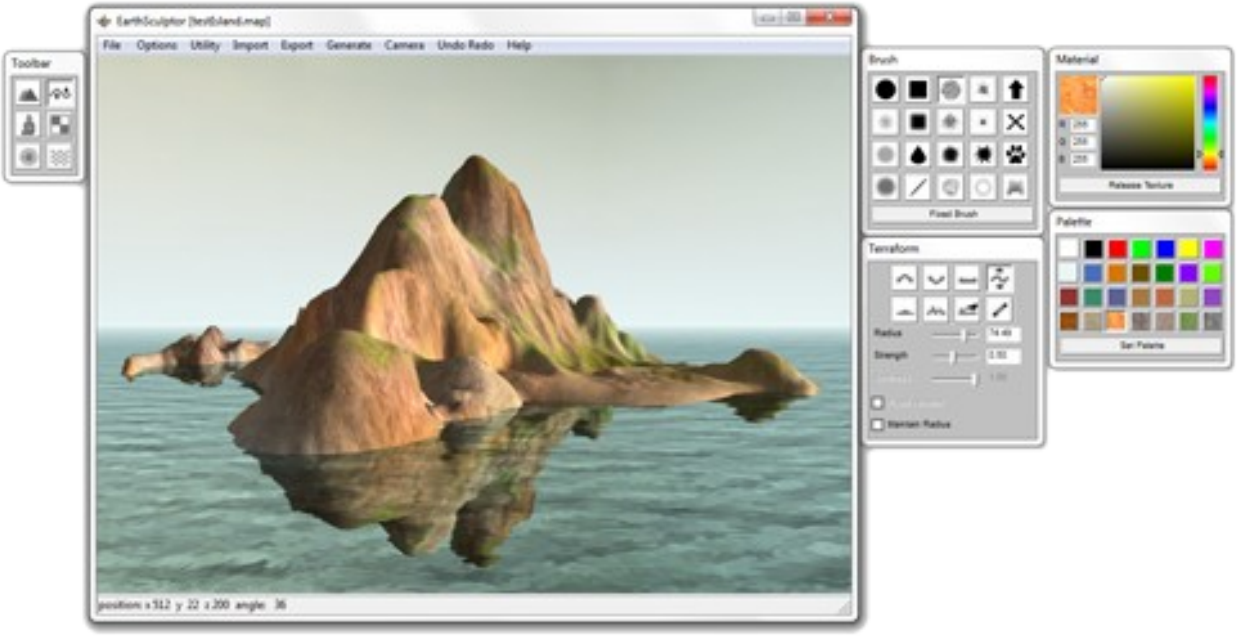
1.2.1 NEM's Mega 3D Terrain Generator [6]

- generátor a editor
- freeware
- MS Windows

Tento program se zaměřuje především na editaci, podporuje práci s texturami a je úzce spjat s hrami používajícími Quake-engine.

1.2.2 Earth Sculptor [7]

- generátor a editor
- komerční, možnost vyzkoušet demo
- MS Windows, OpenGL

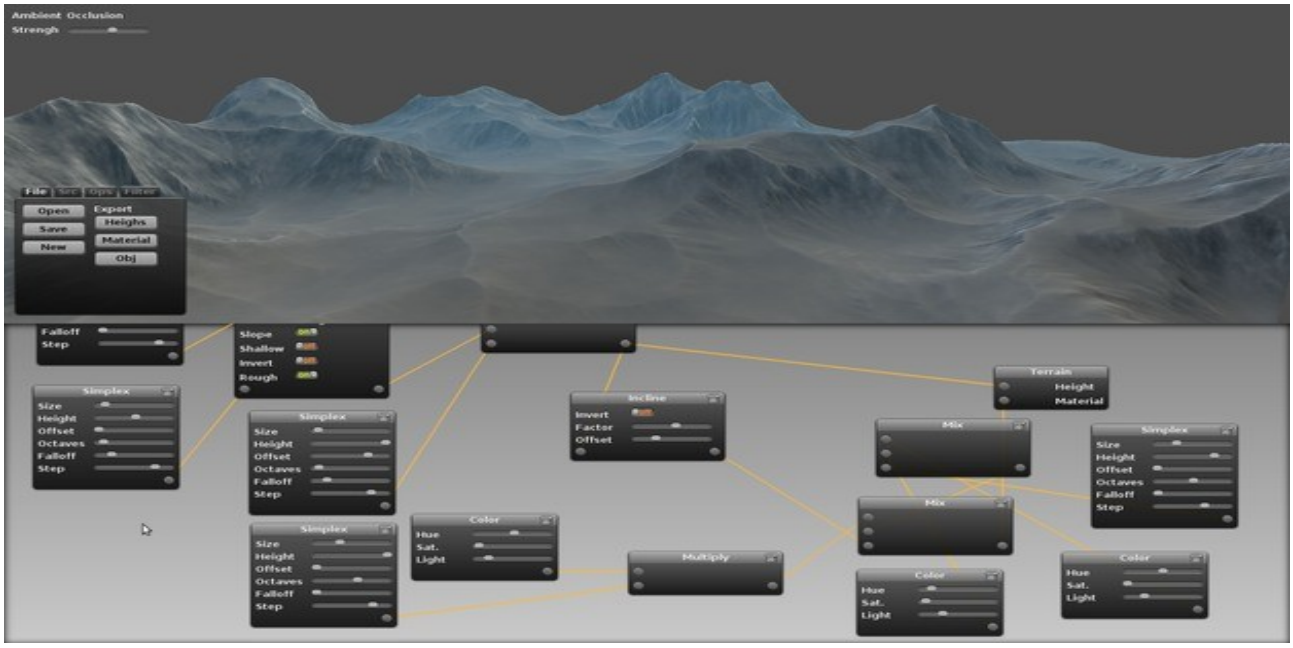


Obrázek 3: Editace reliéfu v programu Earth Sculptor

Earth Sculptor je komerční generátor a editor krajin. Umožňuje kromě běžných funkcí například výpočet stínů a osvětlení, práci s texturami, s vodou atd. Demoverze programu je omezena velikostí mapy, textur atp., je tedy i tak docela dobře použitelná.

1.2.3 Litosphere Terrain Generator [8]

- generátor a editor
- opensource - AGPL
- Linux/Unix, MS Windows



Obrázek 4: Okno programu Litosphere Terrain Generator

Litosphere Terrain Generator mírně vybočuje z předešlých aplikací. Nepracuje klasicky s výškovou mapou, která se nějakým způsobem zobrazí, ale s rovinou, jejíž veškeré vrásnění probíhá na úrovni shaderů. To umožňuje úpravu terénu opravdu v reálném čase.

1.3 Knihovna algoritmů pro generování výškových map

Ačkoliv existuje mnoho nástrojů, které nejrůznějšími způsoby pracují s výškovými mapami a terény, toto téma bude těžko kdy zcela vyčerpáno. V této práci se zaměříme blíže na konkrétní algoritmy generování výškových map, z nichž některé poté zapouzdříme do C++ knihovny, což umožňuje jak jejich pozdější využití v různých aplikacích, tak pohodlné experimentování s algoritmy, či obohacení knihovny o nějaké další.

2 Algoritmy generování výškových map

V této kapitole si blíže popíšeme základní algoritmy generování výškových map a některé zvolené pokročilejší techniky. Tyto metody lze dále kombinovat a upravovat tak, aby bylo dosaženo požadovaných výsledků. Pro přehlednost je v následujícím textu rozdělíme na *algoritmy generování*, a *algoritmy pro filtrování* výškových dat.

V dalším textu jsou volně zaměňovány pojmy *výsledná výšková mapa* a *zvrásněný povrch*, které oba vyjadřují výstupní produkt algoritmů. Ve vztahu k již implementovanému algoritmu nazýváme výslednou entitu konkrétně, avšak věnujeme-li se algoritmům na abstraktní úrovni, význam těchto pojmů splývá.

2.1 Matematický základ

Zajímavým podproblémem týkajícím se všech následujících algoritmů je problém pseudonáhodnosti. Je zřejmé, že v deterministickém prostředí číslicových počítačů nikdy nedosáhneme dokonalé náhodnosti a musíme se spokojit s pseudonáhodnými veličinami, avšak přesto hraje generátor náhodných čísel důležitou roli. Použijeme-li například jiná rozložení pravděpodobnosti [10], než běžné rovnoměrné (které je využíváno v navržené knihovně), může to výrazně ovlivnit výsledný povrch. Podobně jednotlivé algoritmy poskytují více či méně možností, jak pseudonáhodnost korigovat, například v oblasti rozsahů (oboru hodnot generátoru) a jejich změn při iterování. Tyto parametry hrají velkou roli, posuzujeme-li realističnost vygenerovaného terénu – tedy jak moc se podobá reálné krajině. Dosáhnout takové podoby je problém netriviální, jelikož se často ukazuje, že pseudonáhodně vytvořená krajina vykazuje příliš vysoký stupeň homogenity.

2.2 Algoritmy pro generování

Pod tímto pojmem rozumíme algoritmy, které vytváří náhodně zvrásněné povrchy pouze na základě vstupních parametrů, aniž by přitom vycházely z dat, která výšková mapa obsahovala před jejich aplikací.

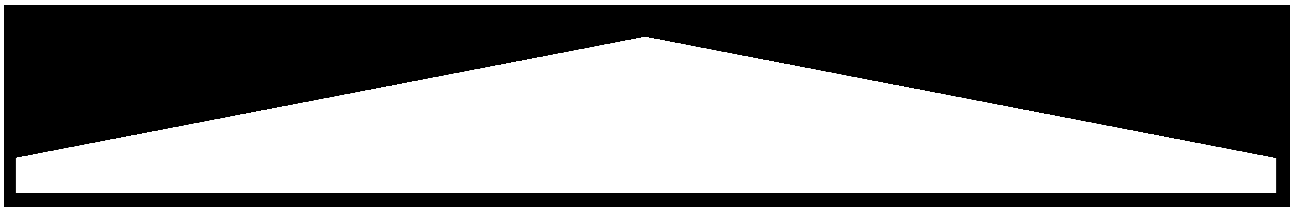
2.2.1 Midpoint (Diamond-square) algoritmus

Jedním ze základních a nejčastěji zmiňovaných algoritmů generujících zvrásněný povrch, je *midpoint algoritmus*, často nazývaný též *diamond-square* [9]. Pro jednoduchost si vezmeme jednorozměrnou variantu algoritmu, jehož vstupem je úsečka a výstupem lomená křivka sestávající se z n úseček, kde n je rovno 2 na počet iterací algoritmu. Tuto

křivku je možno pojímat jako tvar horizontu, což je vidět na obrázcích. V každé iteraci najdeme střed úsečky, který poté posuneme podle osy *y* o náhodnou hodnotu. Tím vzniknou úsečky dvě, na něž se totéž aplikuje v další iteraci. V každé iteraci je třeba snižovat rozsah generátoru náhodných čísel, jelikož se krátí průmět zpracovávané úsečky do osy *x* na polovinu. Polovina je i hraničním koeficientem pro snižování rozsahu generátoru náhodných čísel. Je-li koeficient větší, výsledná křivka je velmi "rozcuchaná".

Pseudokód jednorozměrného midpoint algoritmu by byl následující:

```
do fronty zařaď vstupní úsečku
opakuj v požadovaném počtu iterací
{
      pro každou úsečku
      {
            stanov střed úsečky a rozděl úsečku na dvě
            posuň střed na ose Y o náhodnou hodnotu
      }
      sniž rozsah generátoru náhodných čísel
}
```



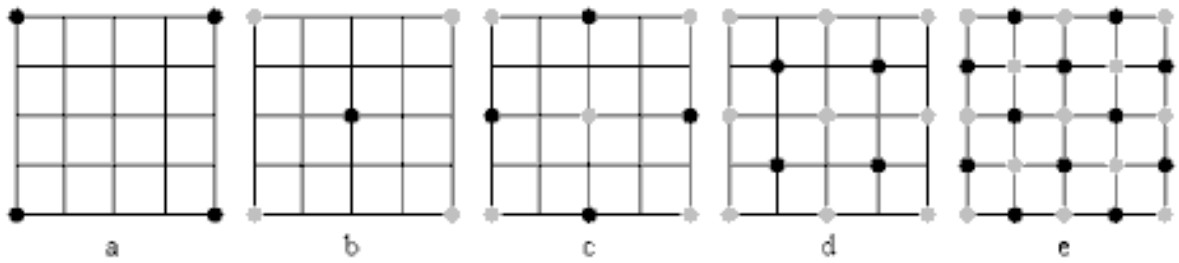
Obrázek 5: Midpoint 1D, 1 iterace

Obrázek 6: Midpoint 1D, 2 iterace

Obrázek 7: Midpoint 1D, 3 iterace

Obrázek 8: Midpoint 1D, 10 iterací

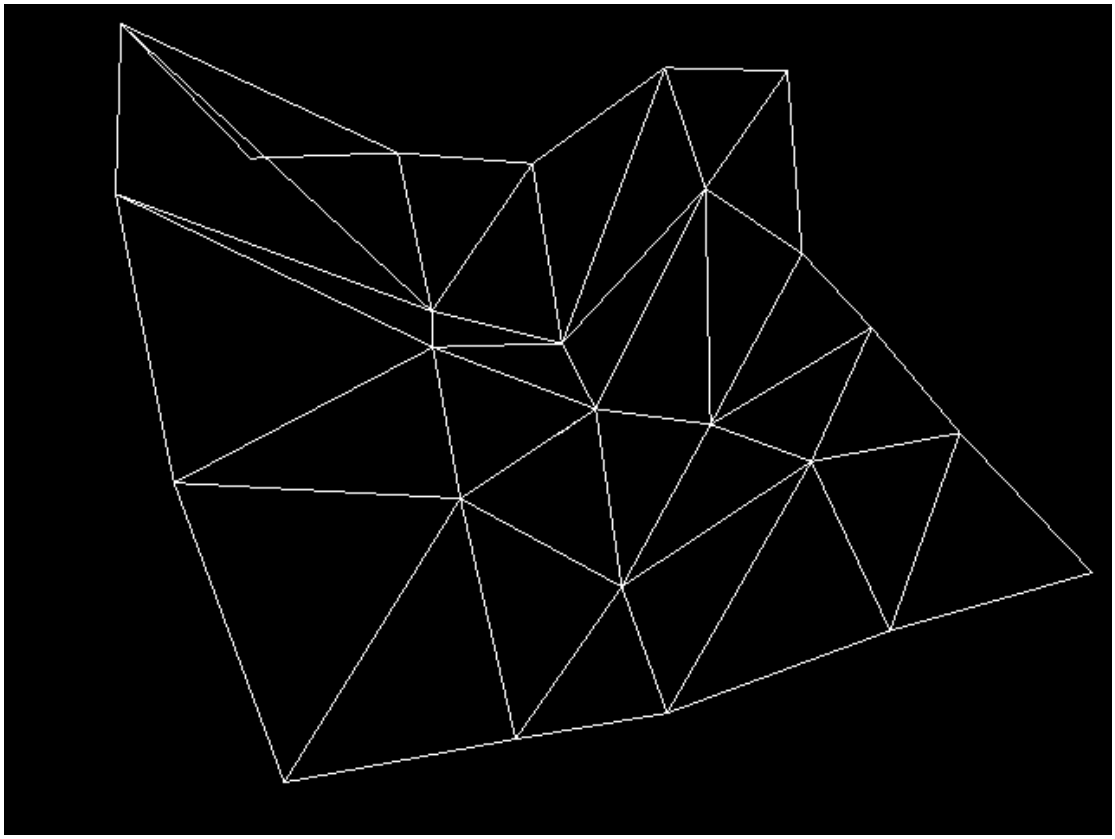
Rozšíříme-li algoritmus o další rozměr, získáme variantu zvanou *diamond-square*. Název vyplývá ze dvou kroků, které se nazývají *diamantový* a *čtvercový*. Zde začínáme čtvercem, nalezneme střed, který dělí čtverec na čtyři další. Středový bod poté posuneme v ose *z* o náhodnou hodnotu. Snížíme rozsah generátoru a iterujeme pro nově vzniklé čtverce.



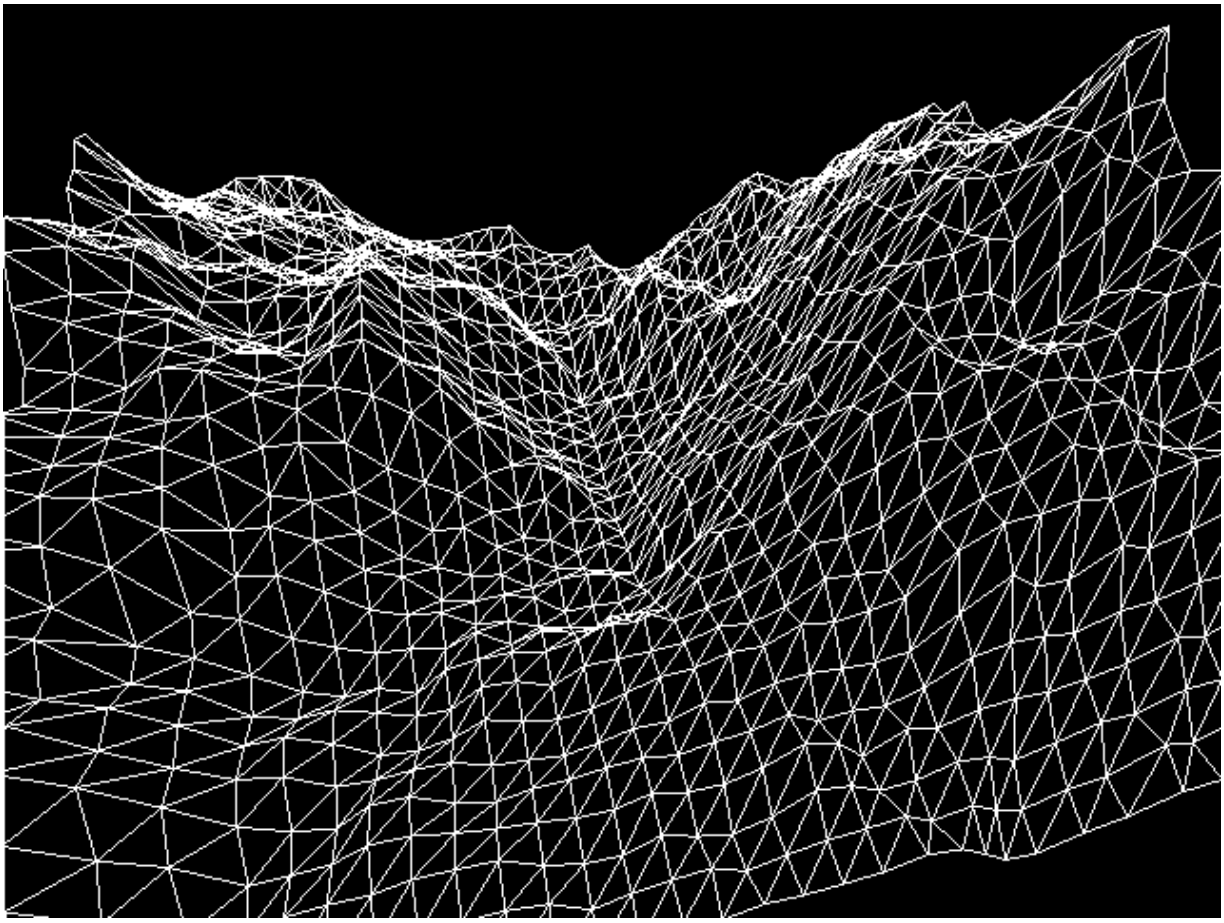
Obrázek 9: Ilustrace čtvercového (a, b) a diamantového (c) kroku algoritmu (převzato z [9])

Pseudokód je zřejmý:

```
do fronty zařaď vstupní čtverec
opakuj v požadovaném počtu iterací
{
      pro každý čtverec
      {
            // čtvercový krok
            najdi střed čtverce a rozděl ho na čtyři
            posuň střed na ose Y o náhodnou hodnotu
            // diamantový krok
            vypočti zbylé čtyři body čtverců (průměry
            sousedních vrcholů velkého čtverce)
      }
}
```



Obrázek 10: Midpoint 2D (Diamond-square), 2 iterace



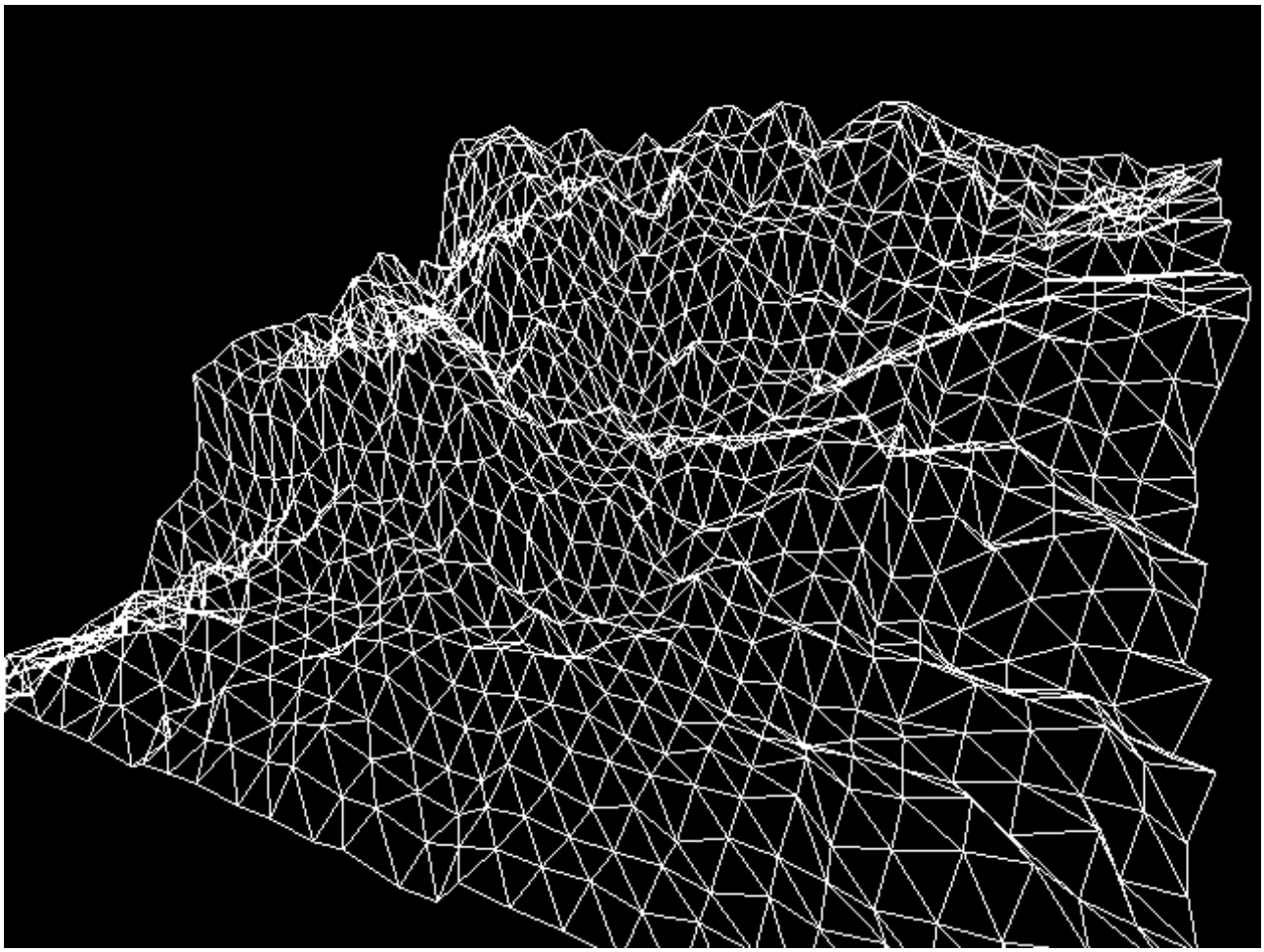
Obrázek 11: Midpoint 2D (Diamond-square), 5 iterací

2.2.2 Dělení roviny

Dělení roviny je velmi jednoduchým algoritmem dosahujícím zajímavých výsledků. Spočívá ve vygenerování náhodné přímky, dělící zadanou rovinu (ta může mít na rozdíl od předchozího algoritmu prakticky libovolný tvar) na dvě poloroviny. Jedna se poté celá zvýší (posune na ose *z*) o hodnotu Δz a druhá o tutéž hodnotu sníží. Provádíme-li tyto kroky dostatečně dlouho, dostaneme náhodně zvrásněný povrch. Problémem tohoto algoritmu je fakt, že v každém cyklu měníme všechny body matice, což může být neefektivní.

Pseudokód algoritmu dělení roviny:

```
opakuj n-krát
{
     urči náhodně body A, B
     pro každý bod K matice
     {
          r = vektorový součin vektorů (A,B) a (A,K)
          pokud je r > 0, zvyš hodnotu v bodě K
          jinak sniž hodnotu v bodě K
     }
}
```



Obrázek 12: Algoritmus dělení roviny, 5000 iterací

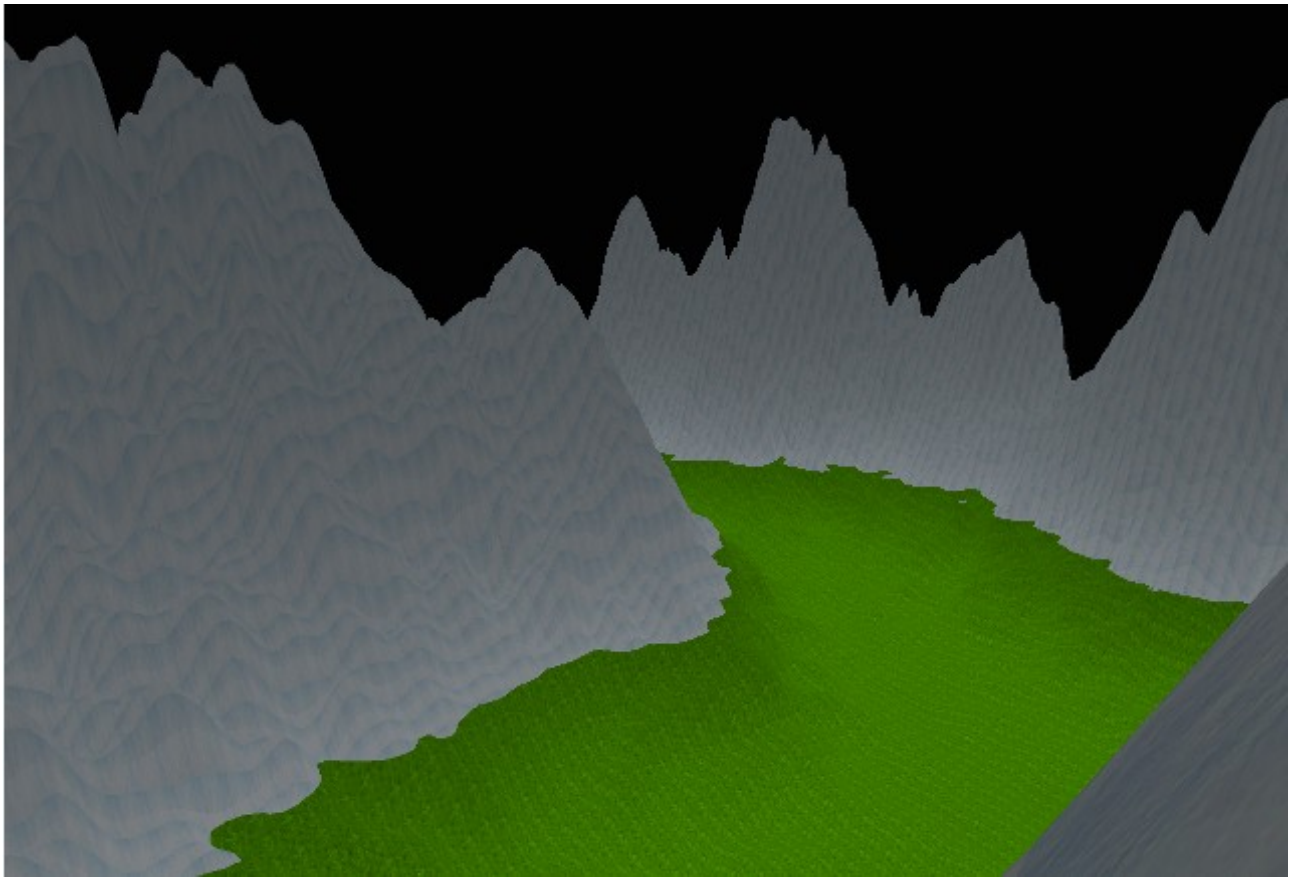
Jak je vidět na obrázku 12, algoritmus dělení roviny produkuje relativně homogenní útvary, což lze očekávat, jelikož za použití rovnoměrného rozložení generátoru náhodných čísel, platí pro každý bod, že pravděpodobnost, že se ocitne na zvyšované nebo na snižované polorovině, je stejná.

2.2.3 Hillgrow

Pokusíme-li se napodobit způsob, jakým vznikají pohoří v reálném světě, dostaneme se ke skupině algoritmů typu *hillgrow*. Tento pojem znamená "růst kopce", což vystihuje jejich společný znak – náhodné rozmístění několika řídících bodů či složitějších útvarů, které jsou základem budoucích kopců. Poté pomocí funkce stanovíme hodnoty okolních bodů výškové mapy, čímž se v závislosti na poloze a hodnotách řídících bodů formuje výsledný zvrásněný povrch. Takovou funkcí může být například suma hodnot Gaussovy funkce [11] pro všechny řídící body. Protože by byl vliv náhodnosti příliš nízký, pokud by se pouze rozmístilo několik bodů o náhodných výškách a zbytek deterministicky dopočítal, je při každém výpočtu Gaussovy funkce náhodně určena hodnota sigma.

Ačkoliv jsou body rozmístěny rovnoměrně, díky Gaussově funkci se vytvoří realistické útvary, jako jsou horské masivy a mezi nimi údolí, které vidíme na obrázku 13. Krajina z obrázku byla vygenerována pomocí třídy HillgrowHMap (viz kapitola 4.1.3) s následujícími parametry:

- rozměry mapy 257 x 257 bodů
- počet vstupních bodů: 15
- 3x aplikovaný box filtr s maticí pro vyhlazení pomocí aritmetického průměru



Obrázek 13: Pohled z hory do údolí krajiny vygenerované pomocí algoritmu Hillgrow

2.3 Filtrování

Na rozdíl od předchozích algoritmů, pracují algoritmy pro filtrování s již vygenerovanými daty, která dále upravují. Jejich účelem jsou jemnější úpravy, například vyhlazení příliš kostrbatých hran nebo doplnění nějakého útvaru.

2.3.1 Box filtr

Jedním z nejznámějších filtrovacích algoritmů je box filtr [12]. Tento algoritmus se používá k mnoha účelům při úpravách obrázků. Jelikož je výšková mapa prakticky tatáž datová struktura jako obrázek ve stupních šedi, lze box filtr aplikovat i na ni.

Vstupem algoritmu je kromě upravované výškové mapy filtrovací matice a požadovaný počet iterací. Pro každý bod výškové mapy se určí nová hodnota vycházející z hodnot okolních bodů. Ta se vypočítá jako vážený součet hodnot osmi sousedních bodů, kde váhy jsou hodnoty filtrovací matice. Součet se pak podělí součtem hodnot filtrovací matice. Obsahuje-li filtrovací matice pouze hodnoty 1, je nová hodnota aritmetickým průměrem sousedních bodů. Možnosti tohoto filtru jsou však mnohem širší. Lze pomocí něj dělat rozmazání obrázku, zostření či detekci hran a další úpravy (viz [12]). Jaký efekt bude algoritmus mít, záleží na hodnotách filtrovací matice.

Pseudokód je následující:

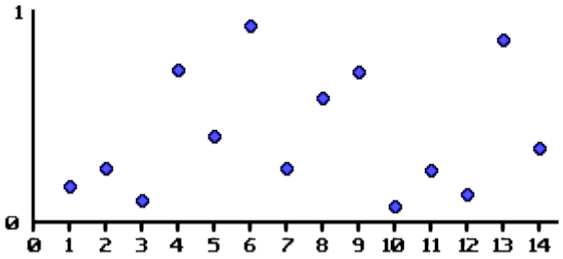
```
jm = suma hodnot ve filtrovací matici
opakuj n-krát
{
      pro každý bod výškové mapy
      {
            p = součet hodnot všech okolních bodů
            vynásobený příslušnými hodnotami filtr. matice
            nová hodnota bodu = p / jm
      }
}
```

2.3.2 Perlinův šum

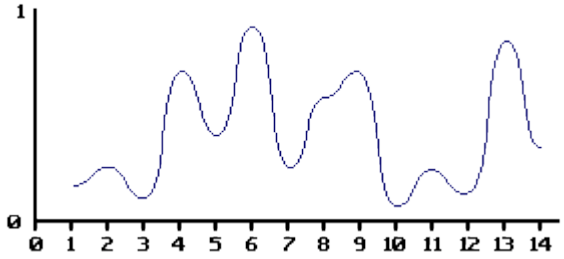
Většina textu a obrázků z této podkapitoly je převzata z [13].

Perlinův šum je metoda, jak generovat reliéf (ať už jedno- či vícerozměrný), který obsahuje *útvary více velikostí* (tento pojem snadno pochopíme, představíme-li si pohoří, které obsahuje hory (velké útvary), kopce, skaliska (menší útvary), kameny (malé útvary)). Naopak, vygenerujeme-li si například průběh funkce sinus, nalezneme v něm pouze útvary jedné velikosti (šířka "kopce" je π a výška 1). Zmenšujeme-li úsek, na který se díváme, křivka se limitně blíží úsečce. Podobně pokud pozorovaný úsek zvětšujeme. Perlinův šum je však odlišné povahy, protože je součtem mnoha funkcí o různých frekvencích a amplitudách.

Základem Perlinova šumu je pseudonáhodná funkce generující šum, který je závislý na zadaném parametru. Podstatné je, že tato funkce je deterministická a se stejným parametrem generuje stejné výsledky. Jelikož potřebujeme spojitý průběh a šumová funkce je diskrétní, je nutno její hodnoty interpolovat. K tomu lze použít různé metody, například lineární, kosinovou nebo kubickou interpolaci (viz zdroj [13]).

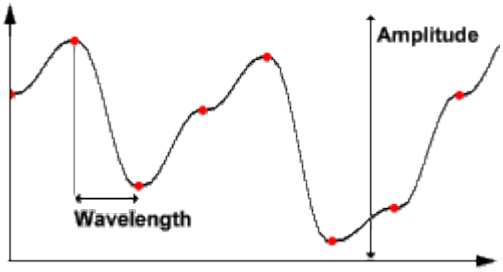


Obrázek 14: Diskrétní šumová funkce



Obrázek 15: Interpolovaná šumová funkce

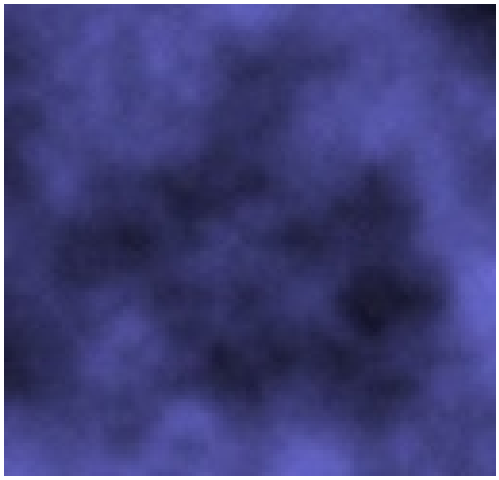
Přestože má šumová funkce pseudonáhodný průběh (viz obrázky 14 a 15), lze u ní definovat pojmy *frekvence*, *amplituda* a *vlnová délka* (viz obrázek 16).



Obrázek 16: Vlnová délka a amplituda u šumové funkce

Vlnová délka je definována jako vzdálenost mezi dvěma interpolovanými body, frekvence pak jako 1/vlnová délka.

Sečteme-li několik šumových funkcí s různými amplitudami a frekvencemi, dostaneme výsledný průběh Perlinova šumu. Často jsou frekvence a amplitudy sčítaných funkcí vzájemně násobky dvou, což je analogické alikvotním tónům [14] v hudbě.



Obrázek 17: Dvourozměrný Perlinův šum

Převedeme-li celý postup do dvou rozměrů, získáme šum podobný obrázku 17. Ten můžeme považovat za samostatnou výškovou mapu nebo použít jako filtr, kdy se hodnoty šumu a původní hodnoty mapy sečtou.

3 Vlastní nové postupy

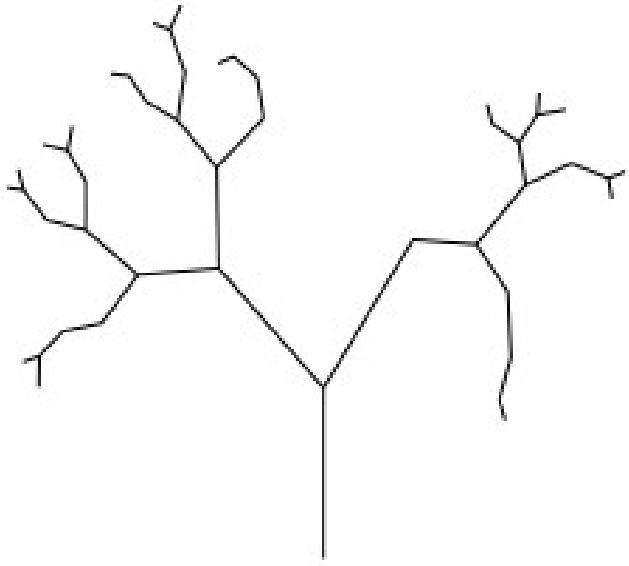
Reálná krajina je tvořena kromě kopců i jinými útvary, například systémem řek a jejich koryt. My se pokusíme vytvořit filtr, který do výškové mapy vyhloubí síť řek s přítoky. K tomuto účelu využijeme možností *L-systémů* [19].

3.1 Co jsou to L-systémy

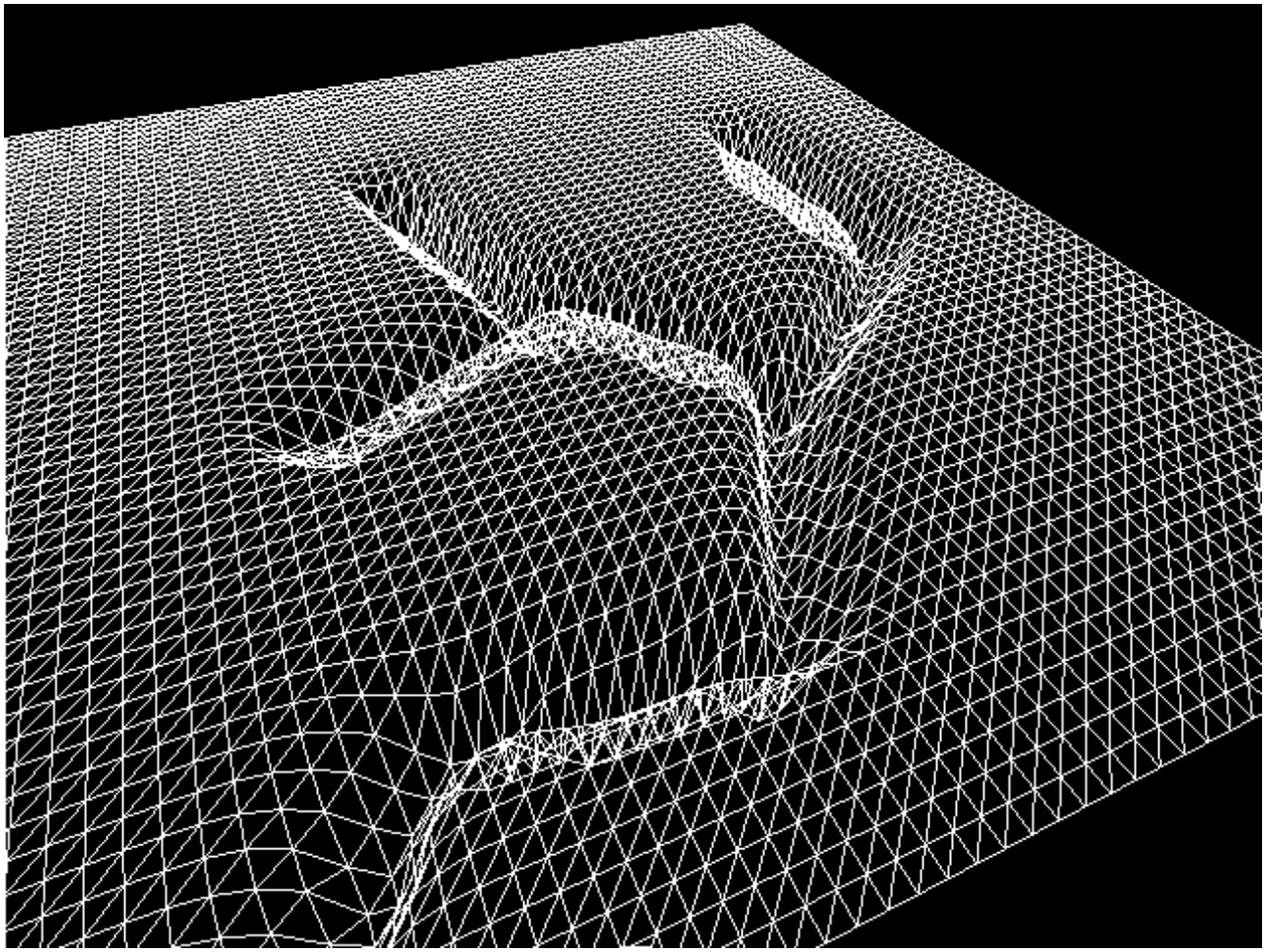
K hlubšímu pochopení problematiky odkazuji na příslušné články ze seriálu o fraktálech na serveru root.cz ([20]) a též na Wikipedii ([19]), odkud byla převzata většina textu této podkapitoly.

L-systém (Lindenmayerův systém) je formální gramatika, jež umožňuje generování fraktálních útvarů. Definuje soubor pravidel, která se rekurzívně aplikují na určitý element, například na úsečku. Aplikací pravidel zpravidla vznikne několik nových elementů, na něž se pravidla následně aplikují znova. Takto postupně vzniká nekonečný fraktální útvar.

My se zaměříme na *stochastické parametrické L-systémy*. Pojem stochastické vyjadřuje vliv náhodnosti na výběr pravidla, které se bude aplikovat. Díky tomu jsme schopni vygenerovat náhodné útvary (například stromy, kde se náhodně určí, zda se v daném bodě bude úsečka dělit nebo ne). *Parametrický L-systém* přiřazuje jednotlivým symbolům parametry, které rozšiřují L-systém o další možnosti. Například vstupní úsečka má délku 1 jednotku a každá další bude o polovinu kratší, čímž nám velikost výsledného útvaru konverguje. Stochastický parametrický L-systém zahrnuje obojí a je vhodným prostředkem pro vygenerování sítě řek a jejich přítoků.



Obrázek 18: Ukázka stochastického L-systému; obrázek převzat z [19]

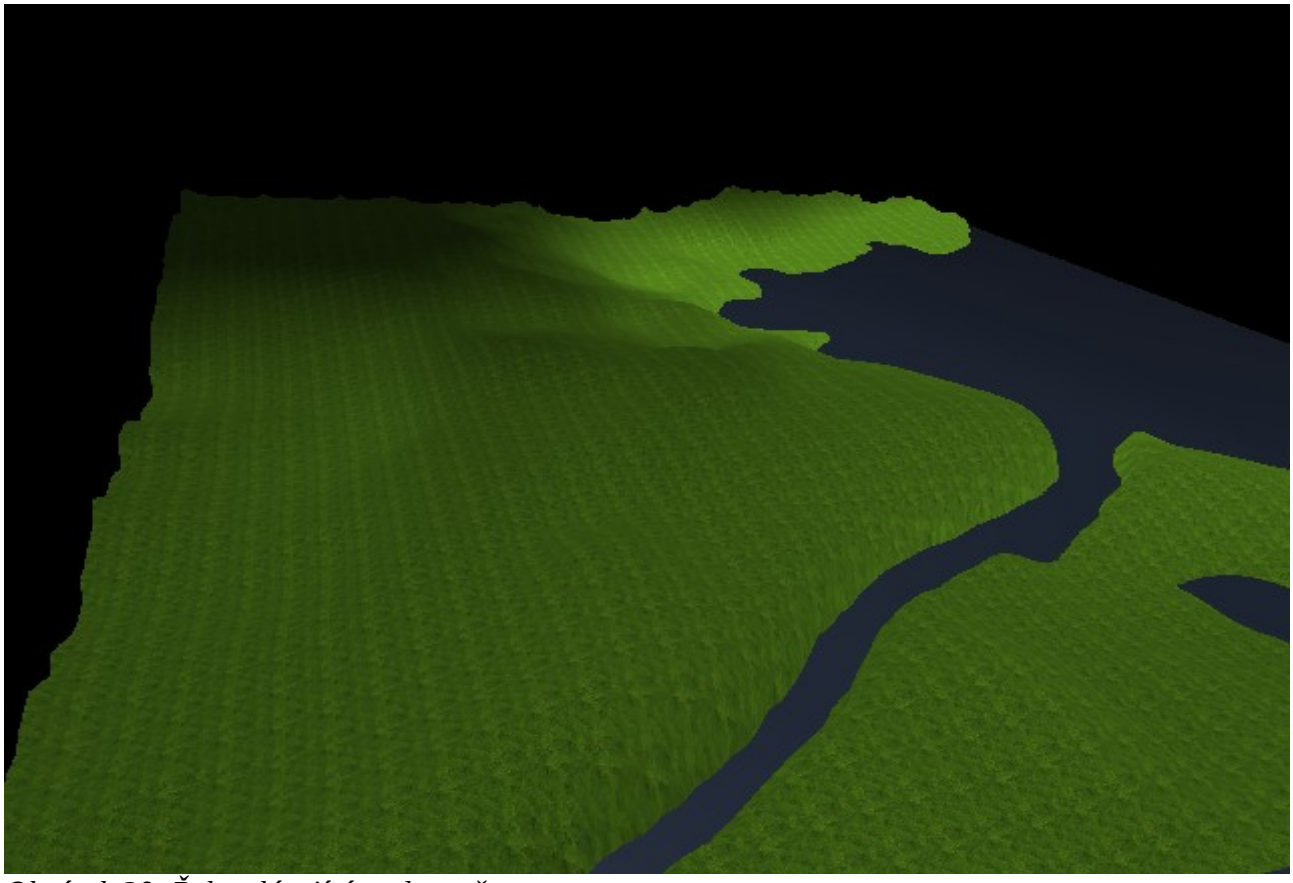


Obrázek 19: Síť řek vygenerovaná pomocí stochastického L-systému zapuštěná v rovině

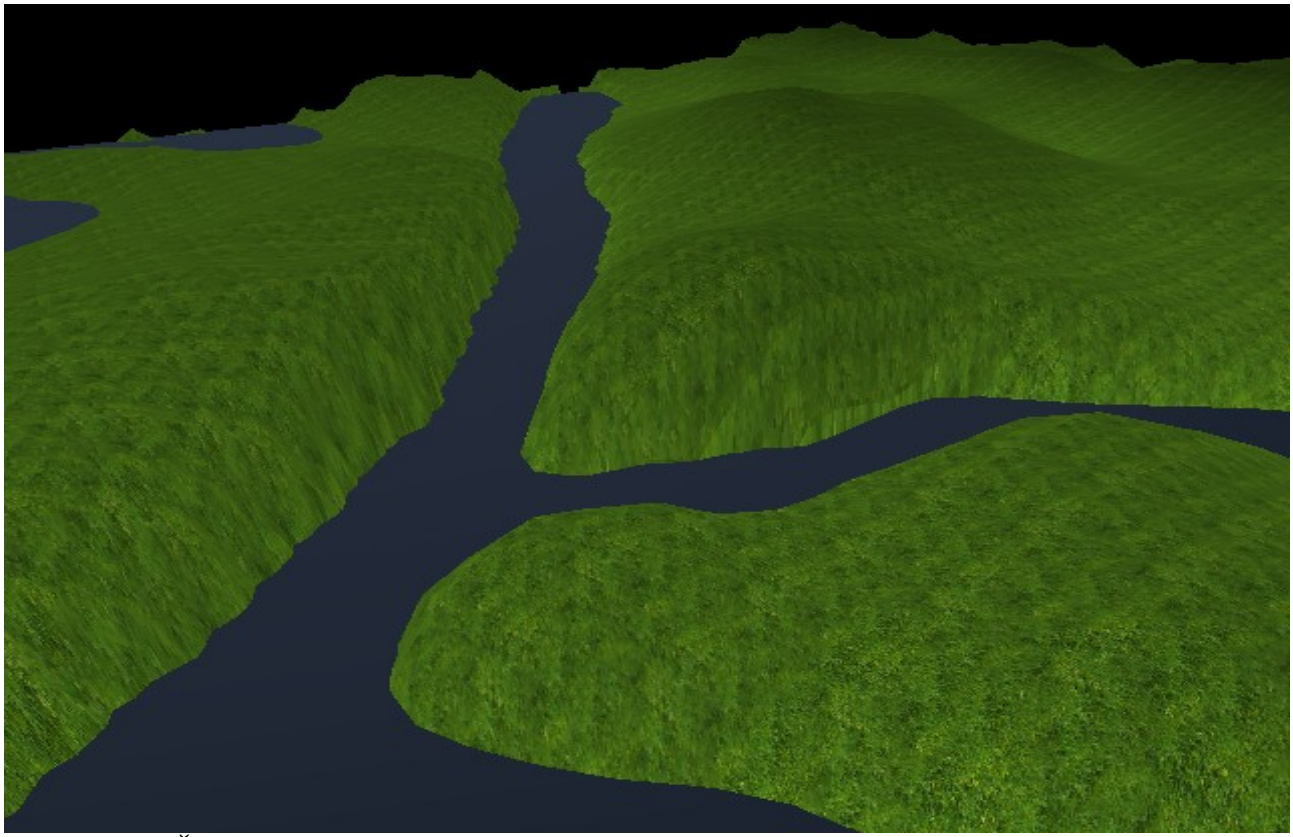
3.2 Aplikace L-systému při generování řek

Řeka se dvěma přítoky na obrázku 19 byla vygenerována pomocí třídy RiversFilter (její bližší popis je v kapitole 4.2.3). Jako vstup zadáme "bod ústí" (výchozí bod, kde se řeka "vylévá" z mapy) a do jaké hloubky chceme až jít. Algoritmus určí náhodný úhel v rozsahu 45° až 135° pod nímž vytvoří úsečku. V jejím koncovém bodě se náhodně rozhodne, zda bude pokračovat v jednom nebo ve dvou směrech a určí se změna jejich úhlů. Takto iterujeme až splníme požadovaný počet iterací. Aby nebyly všechny řeky (bráno od ústí až k prameni každého přítoku) stejně dlouhé (jako je tomu na obrázku), blížíme-li se konci v konkrétní větvi, rozhodneme s určitou pravděpodobností o jejím předčasném ukončení. Protože je řeka jinak široká u pramene než na konci, parametrizujeme jednotlivé body při generování vahou, která klesá směrem od ústí k pramenům a určuje, jak moc velké koryto řeka ve které své části vyhloubí.

Jakmile máme systém řek vygenerován, rasterizujeme jej do výškové mapy. K tomu použijeme Bresenhamův algoritmus [21]. Následně okolní terén vyhladíme pomocí Box filtru. Tyto dva kroky opakujeme tak dlouho, dokud není koryto dostatečně zapuštěno v povrchu.



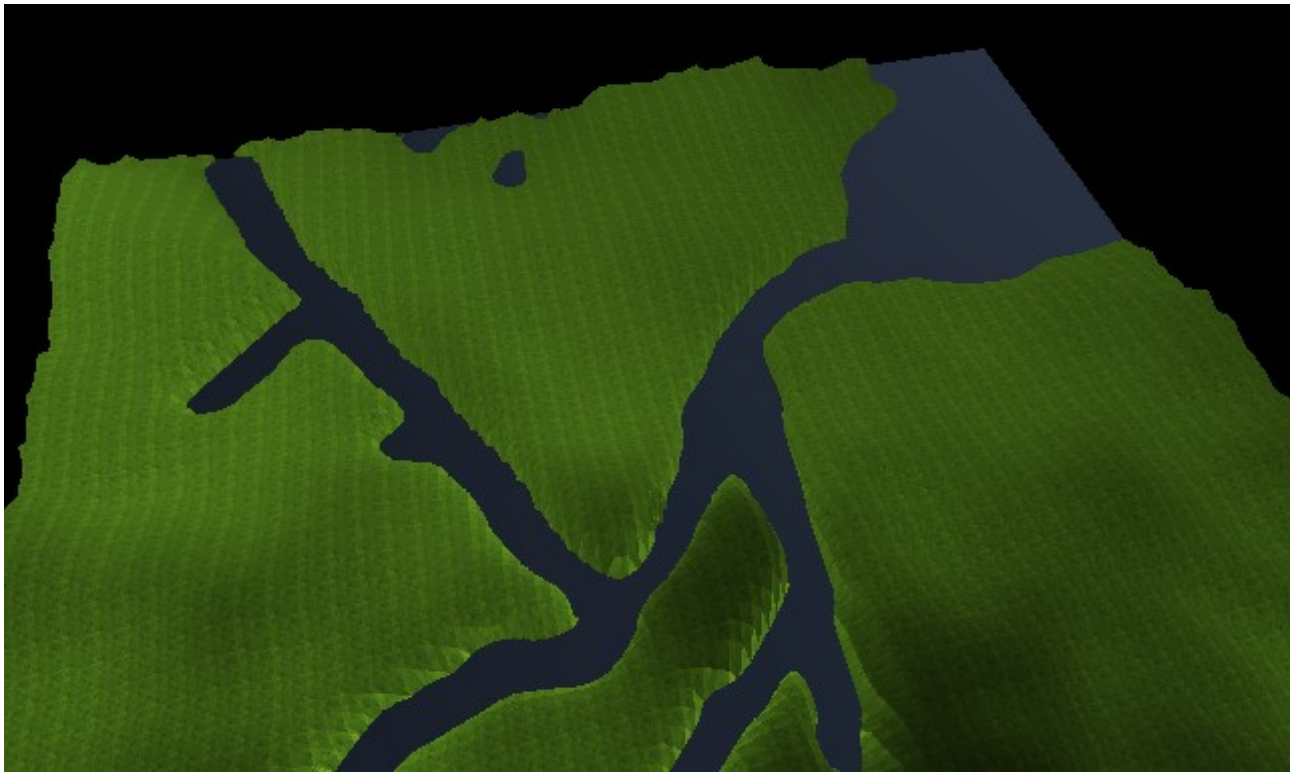
Obrázek 20: Řeka vlévající se do moře



Obrázek 21: Říční koryto v krajině

Na obrázcích 20, 21 a 22 je vygenerované říční koryto začleněno přímo v hotové krajině. Vodní plocha horizontálně protínající krajinu dotváří dojem řeky.

Navržený algoritmus velmi dobře funguje s generátorem založeným na algoritmu dělení plochy (použito v krajinách na obrázcích). Naopak ve spojení s generovacím algoritmem Hillgrow téměř nelze použít. Aby to bylo možné, bylo by nutné algoritmus, který rasterizuje L-systém do výškové mapy, rozšířit o umělou inteligenci, která bude schopna správně rozpoznat výrazné útvary a rasterizaci jim přizpůsobit.

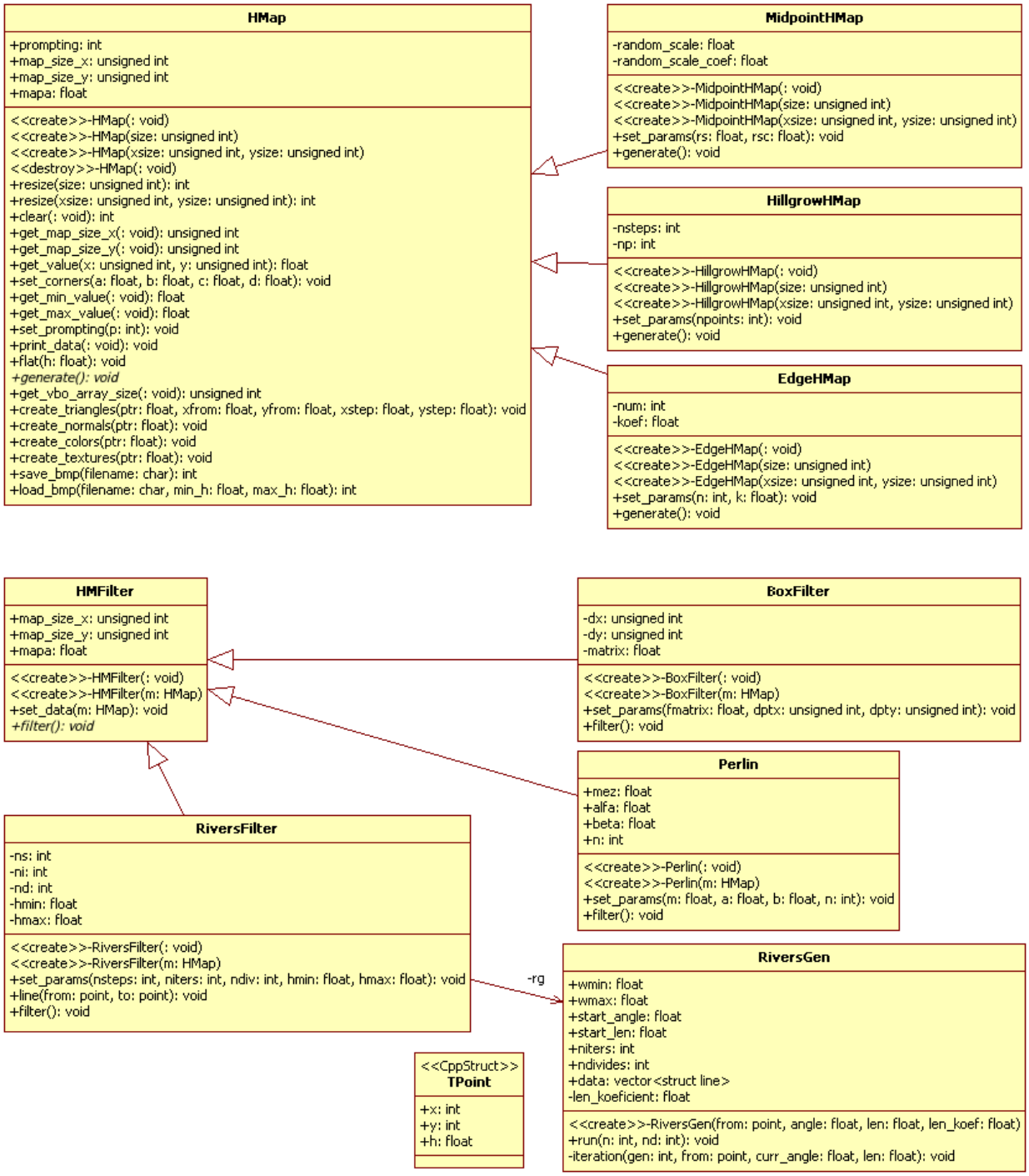


Obrázek 22: Několik říčních ramen v krajině

4 Koncept a implementace knihovny

V rámci práce byla navržena a vytvořena jednoduchá knihovna, implementující některé ze zmíněných algoritmů pro generování a filtrování výškových dat. Tato knihovna může být využita v různých aplikacích, kde je potřeba generovat výškové mapy.

Implementačním jazykem bylo zvoleno C++.



Obrázek 23: Diagram tříd knihovny

4.1 Modul HMap

Jádrem knihovny je abstraktní třída `HMap` (soubory `hmap.h` a `hmap.cpp`), která zapouzdřuje samotná data výškové mapy včetně metod pro změnu jejich velikosti, naplnění inicializačními hodnotami, či vymazání. Z této třídy se dědí třídy `MidpointHMap`, `EdgeHMap` a `HillgrowHMap`, jež představují implementace jednotlivých algoritmů. Každá z těchto tříd dodefinovává čistě virtuální metodu předka `generate()`, vlastní konstruktor a případné další parametry, které podle potřeby jednotlivých algoritmů nastavujeme metodou `set_params()`. Tímto způsobem je možné doplnit do knihovny jakýkoliv další algoritmus.

Dále rodičovská třída obsahuje výstupní rozhraní pro zobrazování pomocí VBO [15] – metody, které výškovou mapu vytriangulují a připraví pole vertexů, normál, texturovacích souřadnic a barev. Aby bylo možno s výškovou mapou pracovat i mimo knihovnu, je zahrnut i export do formátu BMP (metoda `save_bmp(char *filename)`).

Uložené mapy je možné opět načíst metodou `load_bmp(char *filename, float hmin, float hmax)`. Zbývající dva parametry určují rozsah výšek, do nichž se má promítnout rozsah barev z obrázku, v němž jsou hodnoty uloženy jako celá čísla v rozsahu 0-255 (je použita 24bitová barevná hloubka, přičemž výšková data jsou vyjádřena ve stupních šedi). Odlišné parametry formátu BMP metoda neakceptuje.

Protože metoda `load_bmp()` provádí realokaci výškové mapy, je nutné poté aktualizovat ukazatele ve všech připojených filtrech (metoda `HMFilter::set_data()`).

4.1.1 Třída MidpointHMap

Tato třída je implementací Diamond-square algoritmu. Z důvodu optimalizace je algoritmus navržen tak, aby pracoval přímo s výškovou mapou (ačkoliv bychom z jeho popisu očekávali spíše použití fronty, do níž bychom ukládali vygenerované čtverce). Proto je pro korektní výpočet požadováno, aby měla vstupní výšková mapa čtvercový rozměr o straně 2^n+1 bodů. Počet iterací je pak dán hodnotou `n`.

Při výpočtu vycházíme z hodnot v rozích mapy, které je třeba před voláním metody `generate()` nastavit pomocí metody `set_corners(float, float, float, float)`. Má-li mapa požadované rozměry, vygenerované hodnoty se postupně doplňují do výškové mapy, až ji celou zaplní. V opačném případě v ní zůstanou díry.

Máme-li velikostí mapy daný počet iterací, zbývá nám zadat rozsah generátoru pseudonáhodných čísel a koeficient, kterým se bude v každé iteraci rozsah redukovat. K tomu slouží metoda `set_params(float rozsah, float koeficient)`.

4.1.2 Třída EdgeHMap

Třída `EdgeHMap` implementuje algoritmus Dělení roviny. Tento algoritmus je poměrně jednoduchý. Metodou `set_params(int n, float k)` zadáváme pouze požadovaný počet iterací `n` a koeficient `k`, který určuje, o jakou hodnotu se mají poloroviny snižovat nebo zvyšovat.

4.1.3 Třída HillgrowHMap

Tato třída implementuje algoritmus Hillgrow s využitím dvourozměrné Gaussovy funkce. Metodou `set_params(unsigned int n)` zadáme požadovaný počet bodů, které se

pseudonáhodně rozmístí do výškové mapy. Jsou jim přiřazeny pseudonáhodné hodnoty, které určují výšku výsledných kopců. Poté je každému bodu výškové mapy vypočtena hodnota rovnající se součtu hodnot Gaussovy funkce s pseudonáhodnou hodnotou sigma pro všechny vstupní body. Vzniklý terén má příliš ostré hrany, proto je ho třeba ještě vyhladit filtry.

4.2 Modul HMFilter

Modul HMFilter (soubory `hmfilter.h` a `hmfilter.cpp`) je navržen podobným způsobem jako předchozí. Hlavní součástí je stejnojmenná abstraktní třída (`HMFilter`), z níž se dědí třídy implementující konkrétní algoritmy. Ty pak musí dodefinovat čistě virtuální metodu `filter()`. Třídy filtrovacích algoritmů v sobě nezapouzdřují výšková data, ta je nutné předat jednotlivým filtrům v konstruktoru v podobě reference na třídu `HMap`. Z toho plyne, že je třeba mít aspoň jednu instanci třídy `HMap`, k níž můžeme napojit více různých filtrů. Nepředá-li se reference na objekt třídy `HMap` v konstruktoru nebo je třeba nastavení změnit (například po volání `HMap::load_bmp()`), lze to provést pomocí metody `set_data(const HMap &m)`.

4.2.1 Třída Perlin

Zde je implementován filtr pomocí dvourozměrného Perlinova šumu. Metoda `set_params(float m, float alfa, float beta, int n)` nastavuje parametry výpočtu. Parametr `alfa` vyjadřuje změnu amplitudy a parametr `beta` změnu frekvence pro jednotlivé šumové funkce. Parametr `n` určuje počet šumových funkcí, které se budou sčítat, a `m` je maximální amplituda.

4.2.2 Třída BoxFilter

Tato třída implementuje algoritmus Box filtru, jak je popsán v kapitole 2.3.1. Filtrovací matici je třeba předat pomocí metody `set_params(float *matrix, unsigned int x, unsigned int y)`, kde `x` a `y` jsou její rozměry, které musí být liché, aby mohl být nalezen střed. Pointer `matrix` ukazuje na inicializované pole s filtrovací maticí.

4.2.3 Třída RiversFilter

Třída `RiversFilter` implementuje postup popisovaný v kapitole 3.

Metoda `set_params(int k, int i, int s, float hmin, float hmax)` definuje parametry generování. První parametr `k` vyjadřuje počet kroků, kolikrát se má aplikovat box filter. Druhý parametr `i` vyjadřuje počet iterací, neboli z kolika maximálně článků se bude řeka skládat. Parametr `s` stanovuje maximální povolený počet přítoků a poslední dva parametry vyjadřují rozsah výšek, do jakých se má řeka rasterizovat v mapě. Zbylé rozměry jsou určeny automaticky – řeka je vždy umístěna tak, aby vytékala ven z mapy. FIltr se aplikuje stejně jako ostatní metodou `filter()`. Ta volá generování L-systému pomocí modulu LSys (soubory `lsys.cpp` a `lsys.h`), který obsahuje třídu `RiversGen`. Ta na základě parametrů definujících výchozí bod, výchozí úhel, výchozí délku úsečky, požadovaný počet iterací a přítoků, generuje pseudonáhodný L-systém, který je poté vyrasterizován pomocí metody `line()` do výškové mapy a vyhlazen box filtrem.

4.3 Očekávaný způsob použití knihovny

Při návrhu knihovny je třeba brát v potaz způsob, jakým bude knihovna používána a popřípadě i dále vyvíjena. Nabízí se dva protichůdné přístupy, kde stojí proti sobě požadavek jednoduchosti a vysoké variability. Požadavek, aby aplikace, která knihovnu využívá, obsahovala co nejméně kódu (a tudíž práce s knihovnou vyžadovala co nejméně znalostí o její vnitřní struktuře), s sebou nese nevýhodu problematické rozšiřovatelnosti a omezených možností při programování (například ve srovnání se Squirrel skriptovacím jazykem, který z důvodu požadavku široké škály kombinací filtrů a algoritmů využívá zmíněný program GeoGen (viz kapitola 1.2.3)). Tato knihovna je navržena s ohledem na jednoduchost použití.

5 Použití a testování

Knihovna je určena pro použití v jakékoliv uživatelské aplikaci, která vytváří a využívá výškové mapy. Ačkoliv modul HMap obsahuje rozhraní pro napojení na VBO [15], tudíž předpokládá využití OpenGL, není to podmínkou. Data výškové mapy je možné exportovat či dále upravovat, aniž by bylo nutno použít OpenGL.

Pro účel předvedení byla naimplementována aplikace Relief Browser, kde je možné prohlédnout si interaktivně výslednou krajinu ve 3D.

5.1 Popis aplikace

GUI je založeno na knihovně GLUT [16], využívá ke zobrazování Vertex Buffer Object [15] (standard OpenGL). Je primárně určena pro Linux, avšak je možné ji portovat do Windows.

V modulu main jsou instancovány výše zmíněné třídy knihoven HMap a HMFilter, pomocí nichž je vytvořena pseudonáhodná výšková mapa o zadané velikosti. Ta je posléze vytriangulována a předána ke trojrozměrnému zobrazení.

Příklad použití nastiňuje následující kód:

```
// deklarace
MidpointHMap mapa(MAP_SIZE);
Perlin pfilter(mapa);
BoxFilter bfilter(mapa);
float matrix[9] = {1, 1, 1, 1, 1, 1, 1, 1, 1};
...
// nastavíme si hodnoty do rohů
mapa.set_corners(0.0, 0.0, 0.0, 0.0);
// rozsah 0-6 a snižujeme ho o 55%
mapa.set_params(6.0, 0.45);
// generujeme
mapa.generate();
...
// nastavíme a aplikujeme filtry
pfilter.set_params(3.0, 2.0, 2.0, 10);
pfilter.filter();
```

```
...
bfilter.set_params(matrix, 3, 3);
bfilter.filter();
bfilter.filter();
```

Tím máme pomocí algoritmu diamond-square vytvořenou výškovou mapu, na ni jsme aplikovali filtr Perlinovým šumem a dvakrát boxfiltr. Nyní ji můžeme exportovat do formátu BMP pro pozdější použití:

```
mapa.save_bmp("vyskova_mapa.bmp");
```

Pokud chceme mapu vytriangulovat a zobrazit, můžeme použít vestavěné metody, které připraví data pro Vertex Buffer Object:

```
mapa.create_triangles(trojuhelniky, -5, -5, 0.1, 0.1);
mapa.create_normals(normaly);
mapa.create_colors(barvy);
mapa.create_textures(texturovaci_souradnice);
```

Vytvořená pole trojúhelníků, normál, barev a texturovacích souřadnic již standardním způsobem předáme funkci `glBufferSubDataARB()` a výsledný objekt zobrazíme.

5.1.1 Ovládání programu a pohyb ve scéně

Aby bylo možno vygenerovaný zvrásněný povrch intuitivně prohlížet, je v aplikačním modulu zahrnut jednoduchý ovládací model podobný pohledu používanému v 3D FPS hrách (soubor `avatar.h`). Na rozdíl od nich není pohyb omezený gravitací ani hmotou krajiny. Je tedy možné volně procházet povrchem. Program lze ovládat klávesnicí (tabulka kláves je v příloze) a myší. Pohyb myší je pojat jako rozhlížení v horizontálním a vertikálním směru. Změnou směru pohledu je změněn i směr pohybu vpřed a vzad a směr úkroků. Všechny informace o poloze a směru kamery jsou uloženy v objektu `Avatar`, podle jehož obsahu jsou při zobrazování nastaveny projekční matice.

5.2 Zobrazovací modul a texturování

Vygenerovaná krajina je pro realističtější vizuální dojem otexturována několika texturami, jejichž výběr je závislý na výšce daného útvaru. Tím je docíleno zasněžených vrcholků hor a zelených údolí. Aby bylo možné aplikovat více textur na jeden objekt, používá program multitexturing s využitím *programovatelné grafické pipeline* [17]. To umožňuje zasahovat do vykreslovacího procesu nejen na jeho začátku, ale i v různých fázích během něj. K tomu slouží *shadery* (též [17]) – jednoduché programy, které se provádí mezi jednotlivými vykreslovacími kroky. To umožňuje například určit barvu či texturu pro každý bod trojúhelníků, ne jen pro jejich vrcholy, jak tomu bylo u *fixní pipeline* [18].

6 Zhodnocení výsledků

Jak už bylo řečeno, hlavním měřítkem pro hodnocení vygenerovaných reliéfů je subjektivní dojem. Pokud se snažíme o realistické krajinné útvary, budeme porovnávat vygenerovaný reliéf s reálným. V této souvislosti nás bude zajímat již zmíněná homogenita. Reálný reliéf má nízký stupeň homogenity a to ať se na něj díváme prakticky v jakémkoliv měřítku. Pokud si například představíme krajinu státu velikosti naší republiky, můžeme jasně identifikovat hornaté oblasti, tedy místa, kde je výskyt kopců pravděpodobnější než jinde, a nížiny, kde se naopak hory nevyskytují. Naproti tomu, vygenerujeme-li reliéf například pomocí algoritmu dělení plochy, získáme povrch, kde je ve všech místech stejná pravděpodobnost výskytu kopce. Podobně vezmeme-li menší reálné území, například kus pohoří, zjistíme, že obsahuje nejen vrcholy, ale i systém údolí, která vymílají vodní toky atp. Chceme-li toto modelovat, musíme přikročit k pokročilejším technikám. Jednou z nich je například v kapitole 3 popisovaný algoritmus přidávající systém řek do krajiny.

6.1 Výsledky

Přiložené CD obsahuje screenshoty k jednotlivým algoritmům ve větším rozlišení, než by mohl pojmout tento text. Některé nejzajímavější konfigurace je možné vyzkoušet v programu Relief Browser, kde jsou přednastaveny.

7 Závěr

Tato práce se zabývala studiem algoritmů procedurálního generování náhodných terénů. V rámci ní byla vytvořena knihovna implementující vybrané algoritmy, vhodná jak k dalšímu experimentování s algoritmy, tak k využití v aplikacích. Cílem bylo generovat terény, které se podobají reálné krajině, což vedlo k navržení algoritmu hillgrow a filtru, který pomocí L-systémů generuje síť říčních koryt, kterou poté začleňuje do výškové mapy.

V další práci lze navázat na celou řadu témat. Je možné dále zdokonalovat navrženou knihovnu přidáním dalších algoritmů či úpravy stávajících, zejména za účelem věrnější podoby výsledné krajiny krajině skutečné. Spoustu možností k pokračování nabízí generátor řek, kde lze dále experimentovat s parametry a typem navrženého L-systému, nebo vyvinout modul umělé inteligence, který by zapouštěl říční koryta do krajiny s ohledem na charakter krajiny, stávající rozmístění pohoří, údolí a nížin. Zajímavou možností je též integrace vlastních generátorů pseudonáhodných čísel do knihovny, především z důvodu vyzkoušení nerovnoměrných rozložení náhodné veličiny.

8 Literatura

[1] Heightmap. In: *Wikipedia*: *the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2012-05-10]. Dostupné z: http://en.wikipedia.org/wiki/Heightmap

[2] Terraform. GASCH, Robert. [online]. [cit. 2012-05-10]. Dostupné z: http://www.firedrake.org/terraform/

[3] Height Map Editor Home. PRIVANTU, Radu. [online]. [cit. 2012-05-10]. Dostupné z: http://hme.sourceforge.net/

[4] Geogen - Procedural Height Map Generator. [online]. [cit. 2012-05-10]. Dostupné z: http://code.google.com/p/geogen/

[5] Squirrel scripting language. [online]. [cit. 2012-05-13]. Dostupné z: http://www.squirrel-lang.org/ (v době vydání dostupný pouze z Google cache: http://webcache.googleusercontent.com/search?q=cache:ry3c8q_anIEJ:www.squirrel-lang.org/+squirrel+language&cd=1&hl=cs&ct=clnk&gl=cz)

[6] NEM'S Tools: Terrain Generator - About. [online]. [cit. 2012-05-10]. Dostupné z: http://nemesis.thewavelength.net/index.php?p=8

[7] *EarthScupltor*: *Terrain Editor* [online]. [cit. 2012-05-10]. Dostupné z: http://www.earthsculptor.com/

[8] Lithosphere Terrain Generator. BÖSCH, Florian. [online]. [cit. 2012-05-10]. Dostupné z: http://lithosphere.codeflow.org/

[9] Generating Random Fractal Terrain. MARTZ, Paul. [online]. [cit. 2012-05-10]. Dostupné z: [8] http://www.gameprogrammer.com/fractal.html

[10] Rozdělení pravděpodobnosti. In: *Wikipedia*: *the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2012-05-10]. Dostupné z: http://cs.wikipedia.org/wiki/Rozd%C4%9Blen%C3%AD_pravd%C4%9Bpodobnosti

[11] Gaussian function. In: *Wikipedia*: *the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2012-05-10]. Dostupné z: http://en.wikipedia.org/wiki/Gaussian_function

[12] Tech-Algorithm.com: Box Filtering. [online]. [cit. 2012-05-10]. Dostupné z: http://tech-algorithm.com/articles/boxfiltering/

[13] ELIAS, Hugo. Perlin Noise. [online]. [cit. 2012-05-14]. Dostupné z: http://freespace.virgin.net/hugo.elias/models/m_perlin.htm

[14] Alikvotní tón. In: *Wikipedia*: *the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2012-05-10]. Dostupné z: http://cs.wikipedia.org/wiki/Alikvotn%C3%AD_t%C3%B3n

[15] OpenGL Vertex Buffer Object (VBO). HO AHN, Song. [online]. [cit. 2012-05-12]. Dostupné z: http://www.songho.ca/opengl/gl_vbo.html

[16] GLUT: The OpenGL Utility Toolkit. [online]. [cit. 2012-05-12]. Dostupné z: http://www.opengl.org/resources/libraries/glut/

[17] Rendering Pipeline Overview. In: *Wikipedia*: *the free encyclopedia* [online].

San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2012-05-12]. Dostupné z: http://www.opengl.org/wiki/Rendering_Pipeline_Overview

[18] Fixed Vertex Pipeline. [online]. [cit. 2012-05-13]. Dostupné z: http://herakles.zcu.cz/education/Grg/2007/lects/07-grg-02.pdf

[19] L-system. In: *Wikipedia*: *the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2012-05-12]. Dostupné z: http://cs.wikipedia.org/wiki/L-syst%C3%A9m

[20] TIŠNOVSKÝ, Pavel. L-systémy: přírodní objekty i umělé artefakty. [online]. [cit. 2012-05-12]. Dostupné z: http://www.root.cz/clanky/l-systemy-prirodni-objekty-i-umele-artefakty/

[21] Bresenham's line algorithm. In: *Wikipedia*: *the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2012-05-13]. Dostupné z: http://en.wikipedia.org/wiki/Bresenham%27s_line_algorithm

[22] ŽÁRA, Jiří. *Moderní počítačová grafika*. Vyd 1. Brno: Computer Press, 2004, 609 s. ISBN 80-251-0454-0.

[23] RYBA, Jan. *Generování procedurálních terénů na GPU*. 2010. Diplomová práce. FIT VUT v Brně.

9 Seznam příloh

- CD obsahující zdrojový kód knihovny, demonstrační aplikace, popis jejího ovládání a screenshoty vygenerovaných krajin